Chapter.02 데이터 분석 라이브러리-01. Numpy를 사용하는 이유

💡 Numpy : Numerical computing with Python. **수치연산** 및 **벡터 연산**에 최적화된 라이브러리.

Numpy!

NumPy

""" To try the examples in the browser: 1. Type code in the input cell and press Shift + Enter to execute 2.

https://numpy.org/

- 2005년에 만들어졌으며, 100% 오픈소스입니다.
- 최적화된 C code로 구현되어 있어 엄청나게 좋은 성능을 보입니다.
- 파이썬과 다르게 수치 연산의 안정성이 보장되어 있습니다. **(numerical stable)**
- N차원 실수값 연산에 최적화되어 있습니다. == N개의 실수로 이루어진 벡터 연산에 최적화되어 있습니다.

```
# numpy example
import numpy as np

arr = np.array([1, 2, 3])
print(np.linalg.norm(arr)) # print L2 norm of vector (1, 2, 3)
```

Numpy를 사용해야 하는 이유

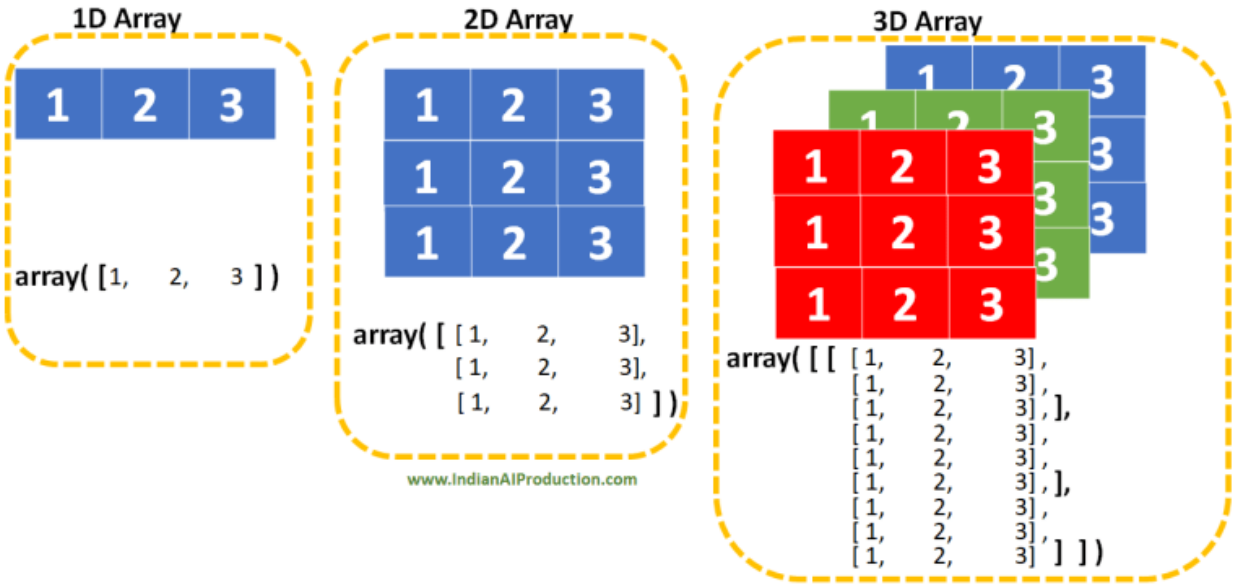
1. 데이터는 벡터로 표현됩니다. 데이터 분석이란 벡터 연산입니다. 그러므로 벡터 연산을 잘 해야 데이터 분석을 잘할 수 있습니다.
2. (native) 파이썬은 수치 연산에 매우 약합니다. 실수값 연산에 오류가 생기면 (numerical error) 원하는 결과를 얻지 못할 수 있습니다. 많은 실수 연산이 요구되는 머신러닝에서 성능 저하로 이어질 수 있습니다.
3. numpy는 벡터 연산을 빠르게 처리하는 것에 최적화되어 있습니다. 파이썬 리스트로 구현했을 때보다 훨씬 더 높은 속도를 보여줍니다.

Hands-on

1. C언어로 코드를 구현하면, 파이썬으로 구현했을 때와 어떤 차이가 있는지 서술해보세요. (CPython)
2. 수치적 안정성(numerical stable)이 없으면 왜 성능 저하를 가져오는지 서술해보세요.
3. Numpy는 어떻게 벡터 연산을 빠르게 처리하는지 서술해보세요.

Chapter.02 데이터 분석 라이브러리-02. Numpy array

💡 Numpy array : numpy에서 사용되는 기본적인 자료구조.



Source : https://indianaiproduction.com/python-numpy-array/

- numpy array는 C언어의 array 구조와 동일한 개념입니다. *(TMI : C array)*
- numpy array는 파이썬 리스트와 비슷한 구조입니다. 하지만, 세부적인 특징이 많이 다릅니다.

< 리스트와 다른 점>

1. 선언한 이후에 크기 변경이 불가능합니다.
2. **모든 원소의 데이터 타입이 동일해야 합니다. (homogeneous array)**

< 리스트와 같은 점>

1. indexing으로 원소를 접근할 수 있습니다.
2. 생성 후 assignment operator를 이용해서 원소의 update가 가능합니다.

- numpy가 제공하는 데이터 타입은 파이썬과 다릅니다.

Numpy type	C type	Description
np.int8	`int8_t`	Byte (-128 to 127)
np.int16	`int16_t`	Integer (-32768 to 32767)
np.int32	`int32_t`	Integer (-2147483648 to 2147483647)
np.int64	`int64_t`	Integer (-9223372036854775808 to 9223372036854775807)
np.uint8	`uint8_t`	Unsigned integer (0 to 255)
np.uint16	`uint16_t`	Unsigned integer (0 to 65535)
np.uint32	`uint32_t`	Unsigned integer (0 to 4294967295)
np.uint64	`uint64_t`	Unsigned integer (0 to 18446744073709551615)
np.intp	`intptr_t`	Integer used for indexing, typically the same as `ssize_t`
np.uintp	`uintptr_t`	Integer large enough to hold a pointer
np.float32	`float`	
np.float64 / *np.float_*	`double`	Note that this matches the precision of the builtin python *float*.
np.complex64	`float complex`	Complex number, represented by two 32-bit floats (real and imaginary components)
np.complex128 / *np.complex_*	`double complex`	Note that this matches the precision of the builtin python *complex*.

- 수치와 관련된 데이터 타입이 대부분입니다.
- 원소의 크기(memory size)를 조절할 수 있으며, 크기에 따라 표현할 수 있는 수치 범위가 정해집니다.
  e.g. np.int8 → 수치 표현에 8 bits를 사용한다 → 00000000 ~ 11111111 → 2^8 (256개) → -128 ~ 127
  e.g. np.float32 → 실수 표현에 32 bits를 사용한다 → exponent, mantissa, sign → single precision